Práctica 3: Buffy the Vampire Slayer. Exceptions and files

Fecha de entrega: 11 de Enero de 2021, 9:00

Objetivo: Manejo de excepciones y tratamiento de ficheros

1. Introducción

En esta práctica se ampliará la funcionalidad del juego en dos aspectos principales:

- Incluir la definición y el tratamiento de excepciones. Durante la ejecución del juego pueden presentarse estados excepcionales que deben ser tratados de forma particular. Además, cada uno de estos estados debe proporcionar al usuario información relevante de por qué se ha llegado a ellos (por ejemplo, errores producidos al procesar un determinado comando). El objetivo último es dotar al programa de mayor robustez, así como de mejorar la interoperabilidad con el usuario.
- Gestionar ficheros para poder grabar y, si fuera el caso, cargar partidas en memoria. Estas tareas requerirían incluir los comandos necesarios para poder realizar tanto la escritura del estado de una partida en un fichero (save) como, llegado el caso, su carga de memoria. La grabación del juego se facilitirá pasando por un comando que genere el estado de la partida (serialize). Con el patrón Command, incluir nuevos comandos para llevar a cabo estas tareas será algo sencillo.

2. Manejo de excepciones

En esta sección se enumerarán las excepciones que deben tratarse durante el juego, se explicará la forma de implementarlas y se mostrarán ejemplos de ejecución.

2.1. Descripción

El tratamiento de excepciones en un lenguaje como Java resulta muy útil para controlar determinadas situaciones que se producen durante la ejecución del juego; por ejemplo,

para mostrar información relevante al usuario sobre lo ocurrido durante el parseo o la ejecución de un comando. En las prácticas anteriores, cada comando invocaba su método correspondiente - generalmente de la clase Game o incluso de la clase Controller - para poder llevar a cabo operaciones sobre el juego. Así si se quería añadir un slayer al tablero, la clase Game debía comprobar si la casilla indicada por el usuario estaba libre y si el jugador disponía de suficientes coins para añadirlo. Si alguna de estas dos condiciones no se daba, la ejecución del comando add devolvía false. De esta forma, con un boolean el juego podía saber si el slayer no se había llegado a añadir, pero a costa de llenar los métodos con mensajes de escritura que indicaban el motivo exacto de lo que había ocurrido (falta de coins o casilla ocupada).

En esta práctica vamos a contar con el manejo de excepciones para realizar esta tarea y ciertos métodos van a poder lanzar y procesar determinadas excepciones para tratar determinadas situaciones durante el juego. En algunos casos, este tratamiento consistirá únicamente en proporcionar un mensaje al usuario, mientras que en otros el tratamiento será más complejo. En esta sección no vamos a ocuparnos de tratar las excepciones relativas a los ficheros, que serán explicadas en la sección siguiente.

En una primera aproximación vamos a tratar dos tipos de excepciones. Por una parte, se va a definir un nuevo tipo de excepciones llamado GameException que será una superclase que recoge dos nuevos tipos de excepciones: CommandParseException y CommandExecuteException. La primera de estas dos sirve para tratar los errores que ocurren al parsear un comando, es decir, aquellos producidos durante la ejecución del método parse(), tales como comando desconocido, número de parámetros incorrecto, tipo de parámetros no válido. La segunda se utiliza para tratar las situaciones de error que se pueden dar al ejecutar el método execute() de un comando; por ejemplo, que un slayer no pueda añadirse al tablero porque el jugador no tiene suficientes coins para hacerlo o porque la casilla donde lo quiere añadir está ocupada. Por otra parte, nos ocuparemos de alguna excepción lanzada por el sistema, es decir, no creada ni lanzada por el usuario. En el juego esto ocurre con la excepción NumberFormatException, que ya se usó en la práctica anterior y que se lanza cuando se produce un error al tratar de transformar un número entero que se encuentra en formato String a su formato habitual int.

2.2. Aspectos generales de la implementación

Una de las principales modificaciones que realizaremos al incluir el manejo de excepciones en el juego consistirá en ampliar la comunicación entre los comandos y el controlador. Continuará existiendo la previa, a través del booleano de retorno del método execute() que indica si se ha actualizado el tablero y, por tanto, este debe ser mostrado. Pero también se contemplará la posibilidad de que se haya producido un error de forma que, controlando el flujo de las excepciones que puedan producirse durante el parseo o la ejecución de un comando, se podrá informar del error al usuario. Además, puesto que ahora se van a tratar las situaciones de error tanto en el procesamiento como en la ejecución de los comandos, los mensajes de error mostrados al usuario podrán ser mucho más descriptivos que en la práctica anterior.

Básicamente, los cambios que se deben realizar son los siguientes:

1. La cabecera del método abstracto parse() de la clase Command pasa a poder lanzar excepciones de tipo CommandParseException:

   ```
   public abstract Command parse(String[] commandWords) throws CommandParseException;
   ```

2. La cabecera del método abstracto execute() de la clase Command pasa a poder lanzar

excepciones de tipo CommandExecuteException:

public abstract boolean execute(Game game) throws CommandExecuteException;

3. La cabecera del método estático parse() de CommandGenerator pasa a poder lanzar excepciones de tipo CommandParseException:

   public static Command parse(String[] commandWords) throws CommandParseException;

   de forma que el método lanza una excepción del tipo

   throw new CommandParseException("[ERROR]: " + unknownCommandMsg);

   en caso de comando desconocido, en lugar de devolver null y esperar a que Controller trate el caso mediante un simple if-then-else.

4. El controlador debe poder capturar, dentro del método run(), las excepciones lanzadas por los dos métodos anteriores, que son subclase de una nueva clase de excepciones GameException

```
public void run() {
    ...
    while (!game.isFinished()) {
        ...
        try { ...  }
        catch (GameException ex) { ... }
    }
}
```

5. Se deben definir nuevas clases (GameException, CommandParseException, CommandExecuteException y algunas más) y lanzar excepciones de estos tipos y tratarlas adecuadamente de forma que el controlador pueda comunicar al usuario los problemas que ocurran. Por ejemplo, el método parseNoParamsCommand() de la clase Command que se proporcionó en la práctica anterior pasa a ser de la siguiente forma:

```
protected Command parseNoParamsCommand(String[] words)
    throws CommandParseException {
        if (matchCommandName(words[0])) {
                if (words.length != 1)
                        throw new CommandParseException("[ERROR]: Command "+
                            name+" :"+incorrectNumberOfArgsMsg);
                else return this;
        }
    return null;
}
```

Haciendo esto así, todos los mensajes se imprimen desde el bucle del método run() del controlador cuyo cuerpo se parecerá al código siguiente:

```
while (!game.isFinished()) {
    if (refreshDisplay) printGame();
        refreshDisplay = false;

    System.out.println(prompt);
    String s = scanner.nextLine();
    String[] parameters = s.toLowerCase().trim().split(" ");
```

```
        System.out.println("[DEBUG] Executing: " + s);
        try {
            Command command = CommandGenerator.parse(parameters);
            refreshDisplay = command.execute(game);
        }
        catch (GameException ex) {
            System.out.format(ex.getMessage() + "%n%n");
        }
    }
```

2.3. Nuevos tipos de excepciones

En el desarrollo de esta práctica debes definir (en algún caso solo incluir y manejar), lanzar y capturar excepciones de, al menos, los siguientes tipos (en la sección siguiente aparecerá alguno más):

- GameException: es la superclase de las excepciones que se deben definir y de la que heredan las subclases CommandParseException y CommandExecuteException.
- CommandParseException: nuevo tipo de excepción lanzada por algún error detectado en el parseo de un comando.
    - NumberFormatException: excepción del sistema lanzada cuando un elemento proporcionado por el jugador debería ser un dato numérico entero y no lo es.
- CommandExecuteException: nuevo tipo de excepción lanzada por algún error detectado en la ejecución de un comando. Son subclases de esta las siguientes:
    - UnvalidPositionException: nuevo tipo de excepción lanzada cuando una posición del tablero proporcionada por el usuario está ocupada o no pertenece al tablero.
    - NotEnoughCoinsException: nuevo tipo de excepción lanzada cuando no es posible realizar alguna acción pedida por el usuario al no tener el jugador suficiente saldo para llevarla a cabo.
    - DraculaIsAliveException: nuevo tipo de excepción lanzada cuando no es posible añadir a Drácula al tablero porque éste ya está presente en el mismo.
    - NoMoreVampiresException: nuevo tipo de excepción lanzada cuando no es posible añadir un vampiro al juego porque no quedan más por salir.

Durante el desarrollo de un juego, éste se encuentra en uno de tres estados: [DEBUG], [ERROR], [GAMEOVER] y de ello se informa en cada ciclo del mismo. De la forma en que se proporciona esta información al usuario y de las situaciones excepcionales que pueden llegar a producirse dan cuenta los siguientes ejemplos:

1. Todo ha ido bien. En el ejemplo, el comando del jugador (b 0 0 20 que consiste en añadir un banco de sangre de coste 20 coins) se ha ejecutado correctamente en modo [DEBUG]:

```
b 0 0 20
[DEBUG] Executing: b 0 0 20
Number of cycles: 1
```

```
Coins: 42
Remaining vampires: 9
Vampires on the board: 1

 ---------------------------------------
|B [20] |       |       |       |       |
 ------- ------- ------- ------- -------
|       |       |       |       |       |
 ------- ------- ------- ------- -------
|       |       |       |       | V [5] |
 ------- ------- ------- ------- -------
|       |       |       |       |       |
 ------- ------- ------- ------- -------
|       |       |       |       |       |
 ------- ------- ------- ------- -------
|       |       |       |       |       |
 ---------------------------------------

Command >
```

2. El juego ha terminado. El estado [GAME OVER] lo refleja. En este juego, a pesar de que el jugador dispone de 1082 coins, hay demasiados vampiros en el tablero, de todos los tipos y con toda su plenitud de vida. El vampiro de más abajo ha alcanzado el lado izquierdo del tablero y, si no se gastan coins, hará que la partida termine y ganarán ellos.

```
Number of cycles: 10
Coins: 1082
Remaining vampires: 4
Vampires on the board: 6
Dracula is alive

 ---------------------------------------
|B [20] |       |       |       |       |
 ------- ------- ------- ------- -------
|       |EV [5] |       |       |       |
 ------- ------- ------- ------- -------
|       | D [5] |       |       | V [5] |
 ------- ------- ------- ------- -------
|       |       |EV [5] |       |       |
 ------- ------- ------- ------- -------
|       |       |       |       |       |
 ------- ------- ------- ------- -------
| V [5] |       |       |       |       |
 ---------------------------------------

...

[GAME OVER] Vampires win!
```

3. Se ha producido un error. En el ejemplo, el comando del jugador (`b 0 7 60` que busca añadir un banco de sangre de coste 60 coins) falla porque la posición (0, 7) solicitada por el jugador, no existe. Observa que tampoco tiene 60 coins para el banco de sangre que solicita, pero lo que se muestra al usuario es el primer error detectado. Tras el intento fallido, que ni ha consumido ciclo ni ha supuesto volver a pintar el tablero, el jugador decide acabar con el vampiro presente en el tablero con un light flash. Pero su nuevo intento también falla. Ahora sí, por falta de saldo. Observa que tanto en un intento como en otro el juego ha pasado del estado [DEBUG] al estado [ERROR] y, en ambos intentos, han aparecido dos mensajes correspondientes a las diferentes excepciones que se han producido:

```
Number of cycles: 1
Coins: 42
Remaining vampires: 9
Vampires on the board: 1

   ---------------------------------------
  |B [20] |       |       |       |       |
   ------- ------- ------- ------- -------
  |       |       |       |       |       |
   ------- ------- ------- ------- -------
  |       |       |       |       | V [5] |
   ------- ------- ------- ------- -------
  |       |       |       |       |       |
   ------- ------- ------- ------- -------
  |       |       |       |       |       |
   ------- ------- ------- ------- -------
  |       |       |       |       |       |
   ---------------------------------------

Command >
b 0 7 60
[DEBUG] Executing: b 0 7 60
[ERROR]: Position (0, 7): Unvalid position
[ERROR]: Failed to add bank

Command >
l
[DEBUG] Executing: l
[ERROR]: Light Flash cost is 50: Not enough coins
[ERROR]: Failed to light flash

Command >
```

Veamos otro ejemplo. El jugador añade a Drácula (`v d 5 2`). Posteriormente intenta añadir un vampiro explosivo (`v e 5 2`) en la misma posición y el intento falla:

```
Command >
v d 5 2
[DEBUG] Executing: v d 5 2
```

```
Number of cycles: 0
Coins: 50
Remaining vampires: 2
Vampires on the board: 1
Dracula is alive

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       | D [5] |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------

Command >
v e 5 2
[DEBUG] Executing: v e 5 2
[ERROR]: Position (5, 2): Unvalid position
[ERROR]: Failed to add this vampire

Command >
```

Más adelante el juego llega a esta situación:

```
[DEBUG] Executing:
Number of cycles: 6
Coins: 70
Remaining vampires: 2
Vampires on the board: 1
Dracula is alive

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       | D [5] |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 7
Coins: 80
Remaining vampires: 1
Vampires on the board: 2
```

```
Dracula is alive

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       | D [5] |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |EV [5] |
   ---------------------------------------------------------------

Command >
```

Observa que el juego ha fallado al añadir, por ciclo consumido, un vampiro normal y un Drácula, pero ha tenido éxito al añadir un vampiro explosivo. Los fracasos pueden haberse debido a que, en los dos casos, la aleatoriedad no ha sido favorable. Supongamos que no, que en el intento de añadir a Drácula la suerte ha sido favorable. No se ha añadido entonces porque ya había un Drácula presente en el tablero. Debes programar este hecho de forma que la aplicación se comporte como cuando el fallo es debido al usuario, es decir, **se lanza una excepción de tipo DraculaIsAliveException, pero esa excepción se trata de tal manera que no se informa del intento fallido al usuario**.

2.4. Ejemplos de ejecución

De guía de cómo pueden ser los mensajes mostrados en respuesta a diversas excepciones lanzadas, vamos a considerar un par de ejemplos. En el primero se muestran errores de parseo y en el segundo, de ejecución.

1. Nivel easy y semilla 1.

```
Buffy the Vampire Slayer 3.0
Random generator initialized with seed: 1
Number of cycles: 0
Coins: 50
Remaining vampires: 3
Vampires on the board: 0

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------

Command >
```

```
help me
[DEBUG] Executing: help me
[ERROR]: Command help :Incorrect number of arguments

Command >
help
[DEBUG] Executing: help
Available commands:
[a]dd <x> <y>: add a slayer in position x, y
[h]elp: show this help
[r]eset: reset game
[e]xit: exit game
[n]one | []: update
[g]arlic : pushes back vampires
[l]ight: kills all the vampires
[b]ank <x> <y> <z>: add a blood bank with cost z in position x, y
[c]oins: add 1000 coins
[v]ampire [<type>] <x> <y>. Type = {""|"D"|"E"}: add a vampire in position

Command >
add 0 vampiro
[DEBUG] Executing: add 0 vampiro
[ERROR]: Unvalid argument for add slayer command, number expected: [a]dd <

Command >
v 0 0
[DEBUG] Executing: v 0 0
Number of cycles: 0
Coins: 50
Remaining vampires: 2
Vampires on the board: 1

   ---------------------------------------------------------------
  | V [5] |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------

Command >
v w 2 2
[DEBUG] Executing: v w 2 2
[ERROR]: Unvalid type: [v]ampire [<type>] <x> <y>. Type = {""|"D"|"E"}

Command >
reseto
```

```
[DEBUG] Executing: reseto
[ERROR]: Unknown command
Command >
reset
[DEBUG] Executing: reset
Number of cycles: 0
Coins: 50
Remaining vampires: 3
Vampires on the board: 0

   ---------------------------------------------------------------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |       |
   ---------------------------------------------------------------

Command >
exit
[DEBUG] Executing: exit
[GAME OVER] Nobody wins...
```

2. Nivel hard y semilla 5.

```
Buffy the Vampire Slayer 3.0
Random generator initialized with seed: 5
Number of cycles: 0
Coins: 50
Remaining vampires: 5
Vampires on the board: 0

   -----------------------------------------------------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   -----------------------------------------------------

Command >
b 6 0 20
[DEBUG] Executing: b 6 0 20
[ERROR]: Position (6, 0): Unvalid position
[ERROR]: Failed to add bank

Command >
b 5 0 30
```

```
[DEBUG] Executing: b 5 0 30
Number of cycles: 1
Coins: 33
Remaining vampires: 4
Vampires on the board: 1

   ---------------------------------------------------------
  |       |       |       |       |       |B [30] |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       | V [5] |
   ---------------------------------------------------------

Command >
a 0 2
[DEBUG] Executing: a 0 2
[ERROR]: Defender cost is 50: Not enough coins
[ERROR]: Failed to add slayer

Command >
b 3 2 40
[DEBUG] Executing: b 3 2 40
[ERROR]: Defender cost is 40: Not enough coins
[ERROR]: Failed to add bank

Command >

[DEBUG] Executing:
Number of cycles: 2
Coins: 36
Remaining vampires: 4
Vampires on the board: 1

   ---------------------------------------------------------
  |       |       |       |       |       |B [30] |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       | V [5] |
   ---------------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 3
Coins: 49
Remaining vampires: 4
Vampires on the board: 1
```

```
   ---------------------------------------------------------
  |       |       |       |       |       |B [30] |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       | V [5] |       |
   ---------------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 4
Coins: 52
Remaining vampires: 3
Vampires on the board: 2
Dracula is alive

   ---------------------------------------------------------
  |       |       |       |       |       |B [30] | D [5] |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       | V [5] |       |
   ---------------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 5
Coins: 65
Remaining vampires: 2
Vampires on the board: 3
Dracula is alive

   ---------------------------------------------------------
  |       |       |       |       |       |       | D [5] |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       | V [5] |       |EV [5] |
   ---------------------------------------------------------

Command >
v d 6 1
[DEBUG] Executing: v d 6 1
[ERROR]: Dracula is already on board
[ERROR]: Failed to add this vampire
```

```
Command >
a 0 2
[DEBUG] Executing: a 0 2
Number of cycles: 6
Coins: 15
Remaining vampires: 1
Vampires on the board: 4
Dracula is alive

   ---------------------------------------------------------
  |       |       |       |       |       | D [5] |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       | V [5] |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       |       |       | V [4] |       |EV [5] |
   ---------------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 7
Coins: 15
Remaining vampires: 1
Vampires on the board: 4
Dracula is alive

   ---------------------------------------------------------
  |       |       |       |       |       | D [5] |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       | V [5] |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       |       | V [3] |       |EV [5] |       |
   ---------------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 8
Coins: 25
Remaining vampires: 1
Vampires on the board: 4
Dracula is alive

   ---------------------------------------------------------
  |       |       |       |       | D [5] |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       | V [5] |       |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       |       | V [2] |       |EV [5] |       |
```

```
   ---------------------------------------------------

Command >

[DEBUG] Executing:
Number of cycles: 9
Coins: 25
Remaining vampires: 0
Vampires on the board: 5
Dracula is alive

   ---------------------------------------------------
  |       |       |       |       | D [5] |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       | V [5] |EV [5] |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       | V [1] |       |EV [5] |       |       |
   ---------------------------------------------------

Command >
g
[DEBUG] Executing: g
Number of cycles: 10
Coins: 25
Remaining vampires: 0
Vampires on the board: 3
Dracula is alive

   ---------------------------------------------------
  |       |       |       |       |       | D [5] |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       | V [5] |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       |       |       |EV [5] |       |       |
   ---------------------------------------------------

Command >
v 4 1
[DEBUG] Executing: v 4 1
[ERROR]: No more remaining vampires left
[ERROR]: Failed to add this vampire

Command >
l
[DEBUG] Executing: l
[ERROR]: Light Flash cost is 50: Not enough coins
[ERROR]: Failed to light flash

Command >
```

```
g
[DEBUG] Executing: g
Number of cycles: 11
Coins: 25
Remaining vampires: 0
Vampires on the board: 2
Dracula is alive

   -------------------------------------------------------
  |       |       |       |       |       |       | D [5] |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       |       |       |       |       |EV [4] |
   -------------------------------------------------------

Command >
g
[DEBUG] Executing: g
Number of cycles: 11
Coins: 15
Remaining vampires: 0
Vampires on the board: 0

   -------------------------------------------------------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  |       |       |       |       |       |       |       |
   ------- ------- ------- ------- ------- ------- -------
  | S [3] |       |       |       |       |       |       |
   -------------------------------------------------------

[GAME OVER] Player wins
```

2.5. Serializar y guardar

En Informática, el término *serialization* (en español, serialización o secuenciación) es la conversión del estado de ejecución de un programa, o de parte de un programa, en un flujo de bytes, habitualmente con el objetivo de guardarlo en un fichero o transmitirlo por la red. El término contrario *deserialization* se refiere al proceso inverso de reconstruir el estado de ejecución de un programa, o de parte de un programa, a partir de un flujo de bytes. Aquí nos interesa producir un flujo de bytes que represente el estado actual del juego –no nos hace falta representar el estado de ejecución completo del programa– con el objetivo de escribir este estado en, y si se diera el caso leer este estado de, un fichero de texto. A menudo se utilizan los términos *stringification*/*destringification* para referirse a la serialización/deserialización cuando se trabaja con un flujo de texto.

En esta práctica vamos a escribir el juego de dos modos: *formatted* y *serialized*. El modo *formatted* es el que hemos visto hasta ahora que consiste en mostrar el juego como un tablero, con una información previa sobre el número de ciclos que se han jugado, la

cantidad de coins que tiene el jugador, etc. El modo *serialized* muestra la información como texto plano. Este texto a su vez nos puede servir para hacer *debug*, pero sobre todo lo utilizaremos para poder salvar el juego en un fichero de manera que, si quisiéramos, pudiéramos recuperarlo para seguir jugando.

Para obtener el modo *serialized* introduciremos un nuevo comando llamado `serialize` (o z en su versión corta; la letra s la reservamos para el comando save) que envía el estado del juego serializado, como texto, a la salida estándar para que se muestre por consola. El formato de esta serialización será el siguiente:

- La primera línea dará información sobre el número de Cycles:: cycleCount
- La siguiente línea dará información sobre el Level:: level.name()
- La siguiente línea dará información sobre el número de Coins:: player.getCoins()
- La siguiente línea dará información sobre el número de Remaining Vampires:: Vampire...
- La siguiente línea dará información sobre el número de Vampires on the Board:: Vampire. ...
- La siguiente línea muestra el encabezamiento Game Object List:
- Después, en cada línea, se da la serialización de cada uno de los objetos del juego que están en el tablero. Para la serialización, de cada uno de estos objetos se guarda lo siguiente:
  - De cada slayer: S;x;y;live
  - De cada regular vampire: V;x;y;live;nextStep. El atributo protegido nextStep de los Vampire sirve para conocer el momento en que se tiene que mover el Vampire
  - De Drácula: D;x;y;live;nextStep
  - De cada explosive vampire: EV;x;y;live;nextStep
  - De cada bank blood: B;x;y;live;cost

Por ejemplo, si el estado del juego es el siguiente:

```
Number of cycles: 7
Coins: 990
Remaining vampires: 1
Vampires on the board: 3
Dracula is alive

 ---------------------------------------------------------
|B [100]|       |       | D [4] |       |       |       |
 ------- ------- ------- ------- ------- ------- -------
|       | S [3] |       |       |       |EV [5] |       |
 ------- ------- ------- ------- ------- ------- -------
|       |       |       | V [4] |       |       |       |
 ---------------------------------------------------------
```

su serialización será:

```
Cycles: 7
Coins: 990
Level: HARD
Remaining Vampires: 1
Vampires on Board: 3

Game Object List:
B;0;0;1;100
V;3;2;4;1
D;3;0;4;1
S;1;1;3
EV;5;1;5;1
```

Observa que el comando `serialize` no cambia el estado del juego. Simplemente vuelca la información en la consola de manera que, tras su ejecución, no es necesario mostrar el tablero de nuevo.

Para implementar este comando se debe definir entonces la clase SerializeCommand. A partir de ella no es difícil definir la clase SaveCommand cuya ejecución es similar a la serialización, pero volcando el resultado en un archivo, al que se le añade un encabezamiento.

En Java existen muchos mecanismos para manejar ficheros. En esta práctica usaremos flujos de caracteres en lugar de flujos de bytes. En particular, recomendamos el uso de BufferedWriter y FileWriter para escribir en un fichero, Además, se recomienda el uso de bloques try-with-resources para el código donde se abre el fichero, capturando IOException en el método execute() de la clase SaveCommand.

Para poder guardar el estado de una partida, deberán tenerse en cuenta las siguientes consideraciones:

- El método parse() de la clase SaveCommand debe lanzar excepciones de tipo CommandParseException si no se proporciona argumento o se proporciona más de uno.
- El método execute() de la clase SaveCommand debe hacer lo siguiente:
  - La clase tiene un atributo String filename para guardar el nombre del archivo que se crea al ejecutar el comando. Para simplificar, vamos a suponer que el nombre proporcionado por el usuario es correcto en varios sentidos. Primero, porque no se va a analizar si los caracteres que componen el nombre conforman un nombre válido de archivo, algo dependiente del sistema operativo. Segundo, porque no se va a comprobar si el programa tiene permisos de escritura en el fichero. Y tercero, porque aunque quisiéramos guardar el juego en un archivo que ya existe, el resultado de la ejecución del comando hará que ese archivo se sobrescriba.
  - Una vez conocido el nombre, el atributo filename de la clase se completará con una extensión .dat. Por ejemplo, si escribimos el comando s datos, la ejecución del juego creará un archivo datos.dat.
  - Se debe crear un FileWriter junto con un BufferedWriter que lo envuelva.
  - Se debe utilizar el métofo serialize() de la clase Game que se empleó en el método execute() de la clase SerializeCommand para generar un string con los datos del juego y después volcarlos en el archivo.

- En caso de haber podido guardar el estado del juego en archivo, con éxito, se debe imprimir el mensaje "Game successfully saved in file <nombre_proporcionado_por_el_usuario>.dat".

Por ejemplo, el contenido del archivo que se obtiene tras hacer save datos sobre el juego de más arriba es el archivo datos.dat siguiente:

```
Buffy the Vampire Slayer v3.0

Cycles: 7
Coins: 990
Level: HARD
Remaining Vampires: 1
Vampires on Board: 3

Game Object List:
B;0;0;1;100
V;3;2;4;1
D;3;0;4;1
S;1;1;3
EV;5;1;5;1
```